НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки
6.050103 “Програмна інженерія’’

на тему: Система захисту даних на базі методу шифрування RSA

Виконав: студент 4 курсу, групи ТВ-51

Ярмошевич Андрій Миколайович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник к.т.н., доц. Крячок Олександр Степанович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент к.т.н., доцент Реуцький М.О. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050103 “Програмна інженерія’’

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту
Ярмошевичу Андрію Миколайовичу

(прізвище, ім’я, по батькові)

1. Тема роботи “Система захисту даних на базі методу шифрування RSA”

керівник роботи к.т.н., доц. Крячок Олександр Степанович

(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201 р.
№ _____

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи файл з формату .json, в якому зберігаються результати серіалізовані дані користувачів
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та засоби захисту інформації з використанням RSA, спроектувати архітектуру системи електронного цифрового підпису, розробити програмне забезпечення, розробити інтерфейс користувача

_____

5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових креслень) 1. Мета та завдання роботи 2. Огляд існуючих рішень 3. Функції системи шифрування RSA 4. Функції генерування електронного підпису 5. Формули розрахунку 6. Використані програмні засоби 7. Висновки

Дата видачі завдання "___"_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2.	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Підготовка матеріалів		
5.	Програмна реалізація системи		
6.	Захист програмного продукту		
7.	Оформлення пояснювальної записки		
8.	Передзахист		
9.	Захист		

Студент _____ Ярмошевич А. М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Крячок О. С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою роботи є створення програми, яка вирішує задачу захисту даних для структур, які зберігають велику кількість документів локально. Програмний комплекс повинен базуватися на методі шифрування RSA. Система передбачає підпис файлів згідно з принципами утворення електронного цифрового підпису (ЕЦП). Програма дає можливість підписати файл унікальним для кожного користувача ключем та перевірити, чи був документ дійсно підписаний користувачем і чи не змінювався він з моменту підпису. Розроблений програмний продукт дозволяє підписати документ необмеженою кількістю користувачів. При розробці передбачено функціонал, що не дозволяє скопіювати, перенести підпис користувача на інший документ.

ABSTRACT

The purpose of the work was to create a program that solves the problem of data protection for structures that store a large number of documents locally. The software package should be based on the RSA encryption method. The system provides for signature files in accordance with the principles of the creation of electronic digital signature (EDS). The program allows you to sign a file unique to each user with a key and verify that the document was actually signed by the user and has not changed since the signature. The developed software allows you to sign the document with an unlimited number of users. The design provides a functional that does not allow you to copy, transfer the signature of the user to another document. A functional is implemented that performs multiplication of large numbers without overflow.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЕЦП – електронний цифровий підпис;

Відкритий ключ ЕЦП – криптографічний ключ, який пов'язаний з секретним (закритим, особистим) ключем спеціальним математичним співвідношенням. Відкритий ключ відомий всім користувачам системи і призначений для перевірки ЕЦП. Відкритий ключ дозволяє визначити автора підпису і достовірність електронного документа, але не дозволяє обчислити секретний ключ.

Закритий ключ ЕЦП – ключ, відомий тільки своєму власнику. Тільки збереження користувачем в таємниці свого закритого ключа гарантує неможливість підробки зловмисником документа і цифрового підпису.

RSA – асиметричний алгоритм шифрування (з відкритим ключем). Базується на складності задачі факторизації великих цілих чисел.

MVC - це патерн проектування веб-додатків, який включає в себе кілька дрібніших шаблонів.

ВСТУП

З активним розвитком інформаційних технологій все більше сфер життя інтегровані або повністю перенесені в онлайн-середовище. Онлайн-системи з кожним днем дозволяють виконувати нові функції і робити зручнішою взаємодію з усталеними структурами. Вже зараз існує багато технологій, які дозволяють абсолютно безпечно проводити фінансові онлайн-транзакції, відправляти та отримувати конфіденційні файли. Завдяки цьому навіть такі відповідальні справи, як підпис документів, укладання контрактів можуть бути інтегровані в онлайн-застосунки. Адаптація процесу підпису документів для мережі зможе забезпечити в майбутньому більш зручний процес документообігу.

На даний момент в законодавчих системах більшості країн, в тому числі і в Україні, існує такий термін як ЕЦП (електронний цифровий підпис). Варто зазначити, що на даний момент ЕЦП в Україні має юридичну силу та може бути рівноправним звичайному підпису або печатці. Також сам принцип дозволяє однозначно визначити, чи був той чи інший документ підписаний конкретною людиною. Тому запропонований функціонал може бути застосованим локально, для захисту електронних даних конкретної корпорації чи структури.

Великі корпорації зберігають велику кількість даних про своїх клієнтів, працівників локально, в електронному вигляді. За допомогою створеної програми можливо забезпечити цілісність цих даних. Таким же чином, як з паперовими документами, достатньо підпису особи, що укладала той чи інший документ. Також програма зможе засвідчити, чи був документ змінений після підпису конкретної особи.

Це дозволить різноманітним структурам тримати документацію в електронному вигляді і забезпечити впевненість в тому, що дані були підписані відповідальним лицем та не змінювались з того часу. При

сприятливих юридичних умовах, застосунок може бути використаний в різноманітних організаціях, взаємодія з якими передбачає підписання договорів. Наприклад, в банках систему електронного підпису можна застосувати для підпису клієнтських договорів і створення нових рахунків дистанційно. За рахунок цього в відділеннях зменшиться кількість відвідувачів і з’явиться можливість повністю автоматизувати процес реєстрації нового клієнта.

Записка містить 4 розділи.

У першому розділі описується постановка задачі системи електронно-цифрового підпису.

У другому розділі описується моделювання алгоритмів, що використовуються для ЕЦП.

У третьому розділі описані засоби реалізації програмної системи

У четвертому розділі дано опис реалізованого програмного продукту та його використання користувачем.

1 АНАЛІЗ СИСТЕМ ЕЛЕКТРОННО – ЦИФРОВОГО ПІДПИСУ ДОКУМЕНТІВ

Метою роботи є полегшення та оптимізація процесів зберігання та використання даних для різноманітних структур, перенесення паперової документації в електронний вид, забезпечення захисту цих даних а також створення функціоналу для підпису документів відповідальними особами та перевірки цих документів іншими користувачами системи.

Розроблена система повинна мати реалізацію алгоритму шифрування RSA, на якому в межах даної роботи базується принцип ЕЦП. Необхідно забезпечити можливість створення та видалення особистих аккаунтів для користувачів, генерування та надання кожному користувачу особистого публічного ключа, що застосовується з асинхронними методами шифрування для забезпечення процедури підпису файлів. Потрібно передбачити можливість підписання документу декількома особами з можливістю перевірити, чи був документ підписаний кожним з ним. Розроблена система повинна відтворювати процес підпису електронної документації для підприємтсва, на якому застосовується. Програма повинна виконувати наступні функції:

- створення нового користувача;
- надання кожному створеному користувачу унікального публічного ключа;
- формування унікального для кожного документу ЕЦП на основі алгоритму

RSA;

- шифрування обраного для підпису документу публічним ключем користувача для формування ЕЦП;
- перевірка на факт підписання обраного документу тим чи іншим

користувачем системи;

- підписання документу декількома користувачами з можливістю визначити

факт підписання документу кожним з них;

- серіалізація та десеріалізація даних користувачів перед збереженням в

системі;

- видалення аккаунту;
- використання захищеного сховища даних, в якому користувацькі дані

представлені в неявному вигляді;

- застосування хеш-функцій для скорочення повідомлення перед шифруванням.

Розроблена система повинна повноцінно мати можливість повноцінно функціонувати онлайн, задля віддаленого доступу, наприклад, з іншого офісу компанії. Це вирішується розміщенням файлу з зашифрованими даними користувачів на сервері, до якого має доступ корпорація. Враховуючи проблему множення великих чисел при шифруванні методом RSA, має бути передбачений функціонал, що виконує множення без переповнення незалежно від довжини множників.

1.1 Аналіз видів шифрування для створення ЕЦП

Алгоритми шифрування поділяють на асиметричні (з відкритим, публічним ключем) та симетричні (з приватним, секретним ключем, одноключові).

Алгоритми шифрування, що називаються симетричними базуються на принципі того, що і відправники, і одержувачі користуються однаковим

ключем. Таємний ключ повинен триматися в секреті і передаватись так, щоб запобігти його перехоплення. В разі якщо таємний ключ захищений, при розшифруванні повідомлення автоматично автентифікація відправника, тому що саме він є власником ключа, за яким можливо зашифрувати інформацію і саме отримувач має ключ, за допомогою якого можна дешифрувати повідомлення. Оскільки відправники та одержувачі є єдиними користувачами, які мають цей симетричний ключ, при спробі скористатись несанкціоновано ключем, буде скомпрометовано взаємодія лише цих двох користувачів. Безпека цього типу системи шифрування залежить від збережності захищеності ключа, що використовується в алгоритмі шифрування, а не від зберігання в секреті алгоритму.

Зазвичай симетричний варіант ЕЦП передбачає присутність в системі третьої сторони – «арбітра», який виступає довіреною стороною обох користувачів. Аутентифікація документу визначається самим факт зашифрування його секретним ключем і передача арбітру. Використовується рідко, тому що ефективних алгоритмів не існує.

Переваги симетричного методу генерування електронного цифрового підпису:

- криптостійкість симетричних схем має високі показники за рахунок стійкості блочних шифрів, що використовується. Їх надійність також достатньо досліджена;

- якщо стійкість шифру недостатня, його легко замінити іншим.

Недостатки симетричних методів для створення ЕЦП:

- необхідно підписувати кожен з бітів інформації, що значно збільшує підпис (він часто буває на порядок довшим за довжину документу);

- створені для підпису ключі можна використати лише один раз, тому що після підпису розсекречується половина секретного ключа.

Будь-які криптосистеми, в основі роботи яких лежить використання закритого ключа, безпосередньо залежать від ступеня секретності цих даних.

Користувач може зберігати ключ на своєму особистому комп'ютері, наприклад, захистивши його паролем.

Але такий варіант має свої недоліки:

- підписувати документи можна тільки на комп'ютері власника ЕЦП;
- збереження даних ЕЦП безпосередньо залежний від захищеності комп'ютера користувача.

Алгоритми шифрування, які називають асиметричними (або шифрування з публічним ключем), відмінно від асиметричних схем, використовують два ключі - секретний ключ (secret key или private key) і відкритий ключ (public key), створені таким чином, що їх послідовне застосування до інформаційного об’єкту не змінює цей об’єкт. Ці ключі різні і не можуть бути отримані один від одного, не дивлячись на те, що вони згенеровані разом. Для зашифрування використовується лише відкритий ключ, для декодування використовується тільки секретний ключ. Розшифровка коду без знання секретного ключа – це надмірно складна задача. Зокрема, задача розрахунку секретного ключа по відомому відкритому ключу є практично нерозв'язною. Основною перевагою шифрування з публічним ключем є простий механізм передачі ключів. При з’єднанні каналом зв'язку передається лише відкритий ключ. Це дозволяє використовувати звичайний канал для цього і виключає потребу спеціального каналу безпеки для пересилання ключа. В асиметричної моделі побудови ЕЦП є свої недостатки:

- асиметричні схеми ЕЦП базуються, як і асиметричне шифрування, на обрахунково складних задачах (задача дискретного логарифмування або задача факторизації);
- криптостійкість алгоритмів шифрування з відкритим ключем поки що не доведена строго математично;
- для покращення криптостійкості необхідно робити довжину ключів більшою.

Саме криптосистеми з відкритими ключами широко застосовуються для генерування електронн-цифрового підпису (ЕЦП). Цифрові підписи, базовані

на асиметричних методах шифрування, дозволяють одержувачу перевірити повідомлення на автентичність джерела інформації (іншими словами, автора інформації), а також перевірити, чи змінилася інформація, в процесі передачі адресату. Тому цифрові підписи є засобом аутентифікації та контролю цілісності даних.

1.2 Юридичні аспекти застосування ЕЦП в Україні

ЕЦП є необхідним атрибутом документа електронного вигляду, який виконує функцію ідентифікації автора та підписувача документа зі сторони інших учасників електронного обігу документів.

ЕЦП у правовому статусі рівнозначний рукописному підпису або печатці, у разі виконання наступних вимог:

- цифровий електронний підпис доповнюється сертифікатом ключа, який використовує перевірені методи підпису;

- в процесі верифікації був використаний посилений сертифікат ключа, який був актуальним на момент формування цифрового підпису;

- приватний ключ абонента відповідає ключу, зазначеному в сертифікаті.

Підпис, сформований за принципами ЕЦП, повинен визнаватися дійсним, без розгляду підстав того, що він має електронну форму або не базується на дійсному сертифікаті ключа. Крім того, закон №852 України передбачає, що користувач, який підписує документ вимушений зберігати особистий секретний ключ захищеним. Це означає, що особа повинна застосовувати ЕЦП лише до електронних документів і не передавати стороннім особам, інакше він буде вважатися скомпрометованим, тобто таким, яким могли користуватись без дозволу.

Закон України передбачає пряму необхідність зберігання персональних кодів ЕЦП у захищеному від самого користувача середовищі. Захищений приватний ключ - це надійний та повністю діючий інструмент цифрового підпису. Програма повинна реалізувати апаратні та програмні засоби, які обмежують збережені дані, а саме значення конфігурації приватного ключа та його копію від несанкціонованого доступу.

1.3 Аналіз вимог до ЕЦП

Алгоритми, що реалізують ЕЦП повинні бути реалізовані так, щоб були дотримані наступні обов’язкові атрибути згенерованого підпису:

- ЕЦП аутентичний, з допомогою програмного забезпечення одержувач документа може довести, що він належить підписнику;

- підпис не може бути помилковим. Тобто підпис мусить свідувати про те, що документ міг підписати той і тільки той користувач, чий автограф знаходиться в документі;

- конкретний ЕЦП не може бути перенесений, підпис є невід’ємною складовою документу і тому не може функціонувати за його рамками, в інших документах;

- при зміні підписаного документу, програма свідчить користувачу про неактуальність перевіреного документу;

- розроблений алгоритм мусить генерувати унікальні публічні ключі з різними за довжиною факторами;

- будь-яка особа, що є власником підпису, може гарантувати, що документ, підписаний його публічним ключем точно свідомить про факт його підписання;

- сучасні програмні засоби не повинні мати змоги відновити секретний ключ користувача, виходячи з вихідних даних програми.

Розвиток документообігу електронного виду та здіймнення електронних платежів, підписання контрактів в межах систем електронної комерції неможливий без розвитку систем забезпечення захисту та цілісності документів. ЕЦП є тією самою технологією, що дає простір для розвитку в цих сферах.

Еволюція ідеї асиметричного шифрування стала найбільш поширеною схемою створення та передачі ЕЦП, тому її застосування є доцільним та оптимальним ріщенням в межах даної роботи.

1.4 Види ЕЦП з юридичною силою

Існуючі ЕЦП можна поділити на три види: проста ЕЦП, посилена некваліфікована ЕЦП та посилена кваліфікована ЕЦП.

У випадку простої системи електронно-цифрового підпису для створення використовуються паролі, коди і інші засоби. Простий електронний цифровий підпис - інструмент підтвердження достовірності електронних даних відправником. Він вважається дійсним при дотриманні наступних умов:

- електронний документ підписаний ЕЦП;

- ключ електронного підпису створений відповідно до вимог інформаційної системи, за допомогою якої проводилася засвідчення і відправка електронних повідомлень відправником.

У нормативних і правових документах, а також договори учасники в обов'язковому порядку визначають основні правила застосування простого електронного цифрового підпису:

- механізм ідентифікації автора підпису в електронному документі;

- обов'язкове дотримання вимог конфіденційності при використанні електронного підпису відповідальними особами;

- виконання вимог закону №852 щодо використання простого електронного цифрового підпису;

- неможливість застосування ЕЦП до секретних державних документів.

Посилена некваліфікована ЕП. Для створення такого підпису використовується

криптографічна програма, що працює на основі ключа електронного цифрового підпису. Посилений некваліфікований підпис дозволяє визначити укладача документа, який його поставилв, і наявність змін в листі після його підписання. Застосування некваліфікованої ЕП дозволяє не використовувати сертифікат ключа електронного цифрового підпису (за умови дотримання вимог законодавства, інших нормативних документів та договорів між відправником і адресатом).

Посилена кваліфікована ЕЦП. Особливість цього різновиду електронного цифрового підпису в наявності спеціального ключа перевірки, що міститься в кваліфікованому сертифікаті. Формування та перевірка посиленою кваліфікованої ЕЦП відбувається за допомогою спеціальних засобів електронного підпису, які відповідають вимогам закону України №852.

Паперові документи з рукописним підписом і електронні документи, на яких стоїть посилений кваліфікована підпис, мають однакову юридичну силу (крім випадків, які визнають виключно рукописний підпис, передбачених в законодавстві). Також законом допускається встановлення в нормативних актах і договорах між відправником і отримувачем додаткових вимог до електронних документів, підписаних посиленим кваліфікованим підписом.

Порівняємо розглянуті види електронного цифрового підпису за аналогією зі знайомими фізичними засобами ідентифікації особистості: Проста ЕП схожа на бейдж - будь-який сторонній чоловік може нею скористатися, тому відповідальність за збереження даних лежить на власнику підпису. Некваліфікована ЕП аналогічна пропуску в компанії, при цьому між сторонами угоди є певний рівень довіри. Кваліфікована ЕП як паспорт -

найважливіший інструмент для ідентифікації, надає можливість користуватися всіма послугами. Відповідно до закону «Про електронний підпис», ЕЦП в Україні створений за закордонними стандартами. Видача сертифіката ключа в іншій державі не може бути причиною невизнання юридичної сили документа, на якому стоїть такий підпис.

1.5 Аналіз актуального стану ЕЦП

Експерти в області використання інфраструктури відкритих ключів пояснили, що інфраструктура відкритих ключів (в міжнародній термінології PKI - інфраструктура відкритих ключів) складається з усіх промислово розвинених країн світу. У той же час, національні ІПК не можуть взаємодіяти один з одним, оскільки навіть у об'єднаній Європі вони не надають необхідного для транскордонного зв'язку юридичного значення. Проблема полягає у використанні національних криптографічних алгоритмів, які є різними для різних країн. Відмінності в алгоритмах не дозволяють повною мірою говорити про взаємну довіру держав у сфері інформаційної безпеки. Крім того, ліквідація національних електронних підписів за межами своїх держав означатиме зміну міжнародного права, наприклад, Гаазької конвенції від 1961 року, яка регулює міждержавне взаємне визнання звичайних паперових документів, прийняття окремої спеціалізованої конвенції про процедуру розпізнавання електронних документів.

Іншою стороною законопроекту є положення про те, що повноваження щодо захисту інформації передаються в інформаційно-телекомунікаційній системі, що використовується під час надання послуг ЕЦП, а також створення особи та забезпечення законності функціонування

належать центру сертифікації ключів. Крім того, очікується, що цей центр буде зобов’язаним:

- своєчасно попереджати абонента та додавати сертифікати відкритого ключа особі, яка заповнила поле про обмеження використання цифрового підпису; здійснювати електронну реєстрацію виданих сертифікатів ключів та реєстрацію заблокованих та анульованих сертифікатів ключів;
- забезпечувати користувачам ключових сертифікатів доступ до іменних

записів протягом дня. Зберігати паперові документи та електронні засоби масової інформації;

- своєчасно інформувати орган з сертифікації або центральний орган з сертифікації про зміну даних, що відтворюються в самому сертифікаті ключа (зміни до статті 8 Закону № 852-IV).

Виходячи з визначення центру сертифікації ключів як частини глобальної служби каталогів, яка відповідає за управління криптографічними ключами користувачів, надання зазначених дозволів дійсно доречно і призначене для забезпечення захисту електронних документів у всіх країнах протягом всього його життєвого циклу.

Повноваження державного нагляду у сфері електронного цифрового підпису,

розробники законопроекту пропонують надати створеному основному органу у системі центральних органів виконавчої влади та забезпечити реалізацію державної політики у сферах специфіки організації зв'язку та захисту інформації, телекомунікацій. Передбачається, що наглядовий орган здійснюватиме аудит центрального органу сертифікації та створення власних сертифікатів ключів.

В законодавчому полі існує багато недоліків з приводу ЕЦП, які заважають повноцінному використанню цієї технології. До цих недоліків можна віднести, наприклад, надмірно звужену сферу застосування. Так, положення Конвенції ООН регламентують тільки правовідносини, що

виникають між комерційними підприємствами, не зачіпаючи при цьому правовідносин, що виникають між держорганами, державою і бізнесом, державою і громадянами, бізнесом та громадянами або між громадянами. Більш того, зі сфери правовідносин, що виникають між комерційними підприємствами, виключено електронну взаємодію в області фінансових і фондових ринків.

З наведених вище причин можна зробити висновок, що технологію ЕЦП важко застосувати для широкого спектру задач у тих галузях, що вимушені регулювати свої процеси через структури держави. Але в випадку приватних корпорацій нічого не заважає застосовувати цю технологію локально, аби не зберігати велику кількість паперових документів в архівах, якими незручно користуватися. За допомогою реалізації ЕЦП компанія, що потребує автентичності даних, зможе бути певна в документах, що зберігаються та відмовитись від паперового документообігу.

Висновки до розділу

Електронний цифровий підпис є найбільш перспективним і широко використовуваним методом захисту електронних документів від підробок і гарантує високу надійність повідомлення. Закони України дозволяють використовувати системи ЕЦП для обміну фінансовими документами та іншими документами, важливими для офісної роботи. Можливе використання коштів, затверджених юридичними службами в системах ЕЦП, що дозволяє обмінюватися документами, підписаними ЕЦП з державними установами.

Основною проблемою з розповсюдженням технологій електронного підпису є відсутність визнаних офіційних або де-факто сертифікаційних органів, які впроваджують інфраструктуру відкритого ключа на основі вітчизняних алгоритмів. Вони не є необхідними для організації системи ЕЦП

окремої структури або компанії. Однак, якщо ви бажаєте в майбутньому вийти на загальнодоступний рівень, сертифікат кореневого центру сертифікації вашої компанії повинен бути виданий державним органом сертифікації.

Електронний цифровий підпис є ефективним рішенням для тих, хто не хоче чекати сотні кілометрів для того, щоб кур'єр, агент компанії чи клієнт прибули для перевірки достовірності отриманої інформації або для підтвердження укладання угоди. Документи можуть бути підписані цифровим способом і доставлені до потрібно місця за лічені секунди. Всі учасники електронного документообігу отримують однакові можливості, незалежно від відстані.

На перший погляд, закон ЕЦП є скоріше технічним стандартом.. Враховуючи, що старий закон ЕЦП був подібний за природою і вважався чимось нішевим, є спокуса не подавати таку важливу правову реформу. Однак резонанс самого закону і проблема електронної ідентифікації в майбутньому набуватимуть ще більшого розповсюдження. На відміну від останнього десятиліття, виявлення клієнтів та ділових партнерів стає все більш важливим для багатьох видів діяльності. По-перше, тому, що бізнес у широкому сенсі все більше залежить від Інтернету, а ділові відносини все більш віддалені. По-друге, тому що компаніям необхідно визначити тих, хто стикається зі зростаючим ретельністю та відповідальністю за невиконання вимог щодо ідентифікації. Тому в ближньому майбутньому використання технологій і практик ЕЦП, може стати необхідним не тільки для банків і компаній з високим рівнем ризику, але і для середнього і малого бізнесу.

2 МОДЕЛЮВАННЯ ПРОГРАМНИХ АЛГОРИТМІВ В КОНТЕКСТІ ЕЦП

Криптосистема RSA (RSA - абревіатура призвіщ засновників) є криптографічним алгоритмом відкритого ключа, який заснований на складній задачі обрахування факторизації або пошуку множників довгих чисел. Алгоритм шифрування названий в честь трьох винахідників - Рона Рівеста (Ron Rivest), Аді Шаміра (Adi Shamir) и Леонарда Едлмана (Leonard Adleman). Алгоритм шифрування RSA став першим методом, придатним для безпосередньо шифрування та ЕЦП. RSA - система відкритого, або публічного, ключа. Алгоритм розроблено в 1977 році. Алгоритм RSA працює так: потрібно два простих числа p і q, до цього ж добуток $n = p * q$ (n - називається модулем). Потім необхідно вибрати таке число e, щоб відповідало наступній умові:

$$1 < e < (p-1) * (q-1) \quad (1.1)$$

А також таке, щоб було взаємно просте (не мало спільних дільників, крім одиниці) з числом $(p-1) * (q-1)$. Потім обчислюється d так, щоб $(e * d - 1)$ ділилось ціло на $(p-1) * (q-1)$, де

(n; e) – відкритий ключ,

(n; d). – секретний або приватний ключ,

«e» - відкритий (публічний) показник,

«d» - закритий (приватний) показник.

Дільники p і q можуть бути знищені або збережені разом з приватним ключем. Якби існували ефективні методи розкладу на множники p і q, вірогідно вдалось би отримати приватний ключ d. Тому надійність криптографічної системи RSA заснована на задачі декомпозиції, яку

практично неможливо вирішити, оскільки в даний час немає ефективного способу пошуку факторів.

Алгоритм RSA в класичній реалізації для шифрування інформації та створення цифрових підписів виглядає таким чином: Допустимо, що Користувач1 (К1) хоче відправити Користувачу2 (К2) повідомлення M. К1 створює зашифрований текст C, взводячи повідомлення M в ступінь e і множачи на модуль n:

$$C = M^e (mod\ n), \tag{1.2}$$

де e і n є відкритим ключем К2. К1 потім посилає C (зашифрований текст) для К2. Для того, щоб розшифрувати отриманий текст, К2 піднесе отриманий зашифрований текст C до степеню d і помножить його на модуль n:

$$M = C^d (mod\ n), \tag{1.3}$$

зв'язок між e та d гарантує, що К2 точно обрахує M. Так як тільки К2 знає d, то тільки він має тільки можливість розшифрувати отримане повідомлення.

Розглянута схема передбачає велику кількість потенційно конфліктних ситуацій.

Розглянутий алгоритм підтверджує, що власник і тільки власник секретного ключа d може створити ЕЦП для деякого документу M, а визначити секретний ключ d по відкритому ключу e не простіше, ніж вирішити задачу факторизації модуля $n = p * q$.

Окрім цього, можливо продемонструвати математично, що результати перевірки ЕЦП буде позитивним тільки в тому випадку, якщо таємний ключ d, що віповідає публічному ключу e, використовується в шифруванні ЕЦП. Виходячи з цього, відкритий ключ e можна назвати ідентифікатором відправника.

В алгоритмі RSA для цифрових підписів існують свої недоліки, не дивлячись на те, що цей алгоритм найбільш розповсюджений для задач створення систем електронно-цифрового підпису. Серед них:

1. Під час обрахунку модуля n, публічного та секретного ключів систем необхідно перевіряти багато умов, а це реалізувати просто практично важко. Не передбачена будь-яка з цих умов дасть простір для підробки цифрового підпису тим користувачем, який помітить таке порушення. В задачі підписання важливих документів з використанням особистих даних користувачів така можливість повинна бути виключена.

2. Для забезпечення криптографічної стійкості підпису RSA, наприклад, на рівні національного стандарту криптографії інформації США (10^{18}) необхідно використовувати цілі числа d, e і n не менше, ніж 2^{512} (близько10^{154}) кожне, а це вимагає високих обчислювальних витрат.

3. ЕЦП RSA загрожує так звана мультиплікативна атака. Іншими словами, алгоритм дозволяє згенерувати, без інформації про таємний ключ d, підписи для документів, в яких хеш-результат повідомлення може бути обрахований за рахунок хешованих результатів вже раніше підписаних документів.

2.1 Швидкодія алгоритму шифрування

Як і в розшифровці і шифруванні, так і при перевірці і створенні підпису, агоритм полягає в основному з піднесення до степеню, яке виконується як ряд множень. У практичних прикладах як правило обирається невелике значення, часто навіть декілька користувачів системи використовують однаковий відкритий показник, але з індивідуальним модулем. Якщо відкритий індикатор однаковий, то деякі обмеження накладаються на основні множники (фактори) модуля. У той же час

шифрування даних відбувається швидше, ніж дешифрування, а перевірка підпису відбувається швидше, ніж підписання. Це оптимально, тому що повідомлення підписується тільки один раз, а от перевірка підпису може бути повторена.

Якщо n - кількість бітів в модулі, то алгоритми, які зазвичай використовуються в RSA, то число кроків, необхідних для виконання операції відкритого ключа - пропорційна другому ступеню n, число кроків для операцій з закритим ключем - третя ступінь n, число кроків для операції створення ключів - четверта ступінь n.

Алгоритми швидкого множення, наприлад, швидке перетворення Фур'є (Fast Fourier Transform, FFT) виконуються в значно меншій кількості кроків. Але при цьому у зв'язку зі складністю реалізації програмного забезпечення, а також із-за того, що з великими розмірами ключів, вони фактично працюють не так швидко, такі методи не стали широко розповсюдженими. Однак якщо говорити про ефективність і продуктивність додатків і обладнання, що реалізують метод RSA, то їх швидкість зростає.

Для процесу шифрування і розшифрування методом RSA існують спеціальне апаратне забезпечення, а саме процесори. Вони реалізовані на «Надвеликих інтегральних схемах» і дають змогу обчислювати операції RSA, що пов'язані з піднесенням великих чисел в велику ступінь по модулю, за короткий час, відносно звичайного обчислення. Однією з найшвидших реалізацій RSA з модулем на 512 біт, базована на надвеликій інтегральній схемі працює з швидкодією 64 Кбіт/с. При цьому апаратна реалізація RSA сама по собі орієнтовно в 1000 разів повільніша, наж апаратна реалізація симетричного алгоритму DES. З майбутнім розвитком технологій ці оцінки швидше за все змінюватимуться, але асиметрична система шифрування RSA ніколи не досягне такої ж швидкодії, як у симетричних криптосистем.

2.2 Вразливості алгоритму RSA

Спроба зламати ЕЦП фактично обмежується зламом алгоритму шифрування. У випадку спроби зламу шифру RSA існує декілька теоретичних та практичних способів. Найбільш ефективним способом є пошук секретного ключа, який відповідає доступному відкритому ключу. Таким чином, зловмисник може підроблювати підписи та прочитати абсолютно всі повідомлення, зашифровані публічним ключем. Така атака може бути виконана шляхом знаходження основних множників загального модуля p і q. На підставі p, q і n (загальний показник) зловмисник може легко обчислити дробовий показник d. Як уже було зазначено, складність полягає в пошуку цих множників. Безпека RSA напряму залежить від факторингу (розкладу на множники), що є майже невиконуваним завданням, для якого немає ефективного рішення.

Насправді, задача відтворення приватного ключа однакова за складністю з проблемою пошуку множників модуля: можна використати d для знаходження співмножників n. А також навпаки, можна застосувати n для знаходження d. Слід зазначити, що вдосконалення комп'ютерних технологій не зменшить криптостійкість від RSA, якщо ключі будуть достатньої довжини. Фактично, модернізація обладнання тільки підвищує надійність криптосистем.

Наступним методом зламу RSA є знаходження методу обчислення кореня ступеня e з $mod\ n$. Так як $C = M^e (mod\ n)$ (1.2), то коренем порядку e з $mod\ n$ є саме повідомлення M. Знайшовши корінь, можна відкривати шифровані повідомлення і підробити підписи навіть без знання закритого ключа. Така атака не є еквівалентом факторингу, але в даний час не існує відомих методів злому RSA таким чином. Однак у особливих випадках, коли багато пов’язаних між собою повідомлень шифрується на основі тих же показників відносно невеликих розмірів, є можливість відкрити повідомлення. Ці атаки є єдиним способом відтворити всі повідомлення, закодовані за допомогою цього шифру RSA.

Існують також інші типи атак, які дозволяють розшифровувати лише одне повідомлення і не дають змоги зловмисникам відкривати інші повідомлення, які зашифровані одним і тим же ключем. Також досліджено та проаналізовано можливість дешифрування тільки якоїсь частини зашифрованого повідомлення.

Найпростішою атакою на конкретне повідомлення є атака з допущенням можливого відкритого тексту. Зловмисник, який має зашифрований текст, повинен допустити, що повідомлення має якийсь конкретний текст, після цього шифрує цей текст публічним шифром одержувача, і порівнює текст, що отримав з існуючим шифрованим текстом. Такий вид атаки можна передбачити способом додавання декількох рандомних бітів в кінці тексту. Інша атака на конкретне повідомлення використовується, коли користувач посилає однакове повідомлення трьом отримувачам відразу, кожен з яких має спільний індикатор e = 3. З цим знанням зловмисник має змогу перехопити цей текст і дешифрувати повідомлення M. Таку атаку можна передбачити знову ж таки шляхом введення декількох випадкових бітів до повідомлення, в цей раз при кожному шифруванні. Крім того, існує декілька атак на зашифрований текст (атаки на окремі відправлені тексти для підробки підписів), в яких зловмисник створює деякий шифрований текст і отримує результуючий відкритий текст, для прикладу, підвівши зареєстрованого користувача до розшифрування підставного повідомлення. Звичайно, є ті, хто не зосереджується не на самій криптосистемі, а на сприйнятливості всієї системи комунікації в цілому.

Такі атаки не можна вважати взломом RSA, так як вони не свідчать про слабкості алгоритму RSA, а про вразливість цієї реалізації. Наприклад, зловмисник може мати секретний ключ, якщо він зберігається без достатньої обережності. Слід зазначити, що для цілковитого захисту недостатньо роботи алгоритму RSA і заходів для захисту безпеки зі сторони математики, тобто використати ключ з достатньою довжиною, оскільки фактично атаки на

незахищені ланки системи взаємодії та генерації ключів RSA мають більший успіх.

2.3 Вибір довжини ключа в алгоритмі RSA

Розмір ключа в алгоритмі RSA пов'язаний з розміром модуля n. Два числа p і q, добуток яких є модулем, повинні бути приблизно однакової довжини, тому що в цьому випадку важче знайти фактори, ніж у випадках, коли довжина чисел істотно відрізняється. Наприклад, в разі якщо ви плануєте згенерувати модуль 768 біт, кожне число повинно бути близько 384 біт. Зауважимо, що якщо два числа надзвичайно близькі один до одного або якщо їхня різниця наближається до певної заданої величини, виникає потенційна загроза безпеці, але ця ймовірність – незначна.

Подані нижче формули демонструють, чому не можна обирати близькі за розміром значення для p і q:

1. Візьмемо $M = (p + q) / 2$

2. Коли $p < q$, маємо $0 \leq M - \sqrt{(n)} \leq (q - p)^{2/8p}$

Оскільки $p = M * \sqrt{(m^2 - n)}$, значення p та q легко можна знайти, якщо різниця p - q досить мала. Ідеальний розмір модуля визначається вимогами безпеки: більший модуль забезпечує більшу безпеку, а також робить повільнішим алгоритм RSA. Розмір модуля в першу чергу базується на важливості даних, що потребують захисту, а також, на основі оцінки потенційних загроз.

Хороший аналіз захисту, що забезпечується конкретною довжиною модуля, наведено в описі дискретного логарифму Рівеста (Riv92a), але те ж саме можна застосувати до алгоритму RSA. У подальшому огляді захисту, що надається ключами RSA різних розмірів, захист аналізується з існуючих методів факторизації та перспектив їх розвитку, а також вивчає можливість

залучення значних ресурсів обробки даних через обчислювальну техніку. Оцінка, проведена в 1997 році, показала, що 512-бітний ключ RSA може бути відкритий (факторингом) за $ 1 000 000 і вісім місяців. У 2008 році 512-бітний ключ був відкритий за місяць, що означає, що 512-бітові ключі більше не забезпечують достатньої безпеки, за винятком дуже короткочасних завдань безпеки.

В даний час лабораторія RSA рекомендує ключі розміром 1024 біта для звичайних завдань і 2048 біт для особливо важливих завдань. Менш цінну інформацію можна безпечно зашифрувати за допомогою ключа 768 біт, оскільки цей ключ завжди недоступний для всіх відомих інвазивних алгоритмів. Щоб оцінити рівень безпеки в різних форматах ключів, можна скористатися моделлю, запропонованою Lenstra і Verheul або скристатися авторитетними рекомендаціями.

Як правило, ключ для окремого користувача має певний очікуваний термін служби, який закінчується через певний час, наприклад, через рік. Це дозволяє здійснювати регулярну заміну ключа та забезпечити необхідний рівень безпеки. Після закінчення терміну дії ключа користувач повинен створити новий ключ, переконавшись, що налаштування криптографічної системи залишаються тими ж, особливо у випадку, якщо система використовує ключі близької один до одного довжини. Звичайно, заміна ключа не захищає від атак на повідомлення, зашифровані минулим ключем, але для цього розмір ключа повинен бути обраний відповідно до очікуваної тривалості релевантності даних та враховувати перспективи розвитку технологій.

Можливість заміни ключів дозволяє керувати криптографічною системою у відповідності з ключовими правилами, що регулярно публікуються лабораторією RSA. Користувачі повинні брати до уваги, що оцінений час вторгнення для системи RSA є лише середнім значенням, а масова атака на тисячі модулів у деяких випадках може дати позитивний результат за відносно короткий час. Хоча надійність кожного ключа і

залишається високою при достатній довжині, деякі методи факторизації завжди залишають зловмисникові невеликий шанс швидко знайти ключ.

Стосовно ускладнення злому, пов'язаних зі збільшенням розміру ключа: при зміні величини модуля в два рази, час операцій збільшується в середньому в чотири рази, коли виконується операція відкритого ключа (шифрування та перевірка підпису), а при виконанні операції закритого ключа (дешифрування та підписання) - вісім разів. Різниця між часом роботи між відкритими ключами та секретними ключами пояснюється тим, що відкритий показник може залишатися незмінним, а модуль збільшується, а довжина приватного індикатора збільшується пропорційно до збільшення довжини ключа. Час, необхідний для створення ключів, при вдвічі довшому модулі зростає в 16 разів, але це операція, що виконується рідко і, як наслідок, практично не впливає на загальну продуктивність.

2.4 Криптоаналіз алгоритму RSA

Трьома можливими підходами до криптоанализу алгоритму RSA є наступні:

1. простий перебір. Передбачає перевірку всіх можливих особистих ключів;

2. Математичний аналіз. Існує кілька підходів такого роду, але всі вони по суті еквівалентні знаходженню множників твори двох простих чисел.

3. Аналіз часових витрат. Спирається на аналіз часу виконання алгоритму дешифрування.

Захист проти простого перебору в разі RSA залишається тим же, що і для всіх інших криптосистем - використання великого простору ключів. З цієї точки зору, чим більше бітів e і d, тим краще. Однак через складність

обчислень як при генеруванні ключів, так і при шифруванні/дешифруванні, чим більшим виявляється розмір ключа, тим повільніше працює система.

Для відомих сьогодні алгоритмів проблема визначення d за вихідними даними вимагає, по крайней мере, таких же витрат часу, як і проблема розкладання на множники. Отже, витрати на вирішення завдання розкладання на множники можна використовувати як еталон при оцінці ступеня захищеності RSA.

Для великих модулів n з великими простими множниками, розкладання на множники є серйозною проблемою, але не настільки, як потрібно для забезпечення високої криптостійкості алгоритму RSA.

Для доказу того, що захист криптографічного алгоритму виявляється непростим завданням, як не можна краще підійде опис методу аналізу тимчасових витрат. В даному випадку противник отримує можливість визначити особистий ключ, аналізуючи витрати часу, які потрібні комп'ютеру для розшифровки повідомлень. Аналіз тимчасових витрат можна використовувати не тільки в разі RSA, але і з іншими системами шифрування з відкритим ключем. Атаки такого роду виявляються небезпечними з двох причин:

- вони можуть вестися з найнесподіваніших напрямків;

- вони можуть передбачати аналіз тільки шифрованого тексту.

Аналіз тимчасових витрат частково аналогічний підходу зломщика, відгадували комбінацію замку сейфа, спостерігаючи за тим, скільки часу потрібно власнику для того, щоб повернути пристрій замка від цифри до цифри.

Суть цього підходу можна пояснити на прикладі алгоритму зведення в ступінь в арифметиці класів відрахувань при довічним розкладанні показника ступеня n. В даному алгоритмі підведення до степеню виконується біт за бітом, з одним множенням на кожній ітерації і додатковим множенням, виконуваних для кожного біта, рівного 1.

Припустимо, що система шифрування використовує функцію множення в класах відрахувань, яка виконується дуже швидко майже у всіх випадках, крім невеликого їх числа, коли множення займає значно більше часу. Аналіз проводиться біт за бітом, починаючи з першого зліва біта. Припустимо, що перші k бітів вже відомі. (Щоб отримати значення показника ступеня, слід почати з k = 0 і повторювати процедуру аналізу до тих пір, поки не стануть відомими всі біти показника). Для наявного шифрованого тексту противник може виконати перші j ітерацій. На наступному етапі виконання операції залежить від значення невідомого біта показника ступеня. Якщо цей біт дорівнює 1, то виконується оператор множення вже наявного добутку d на e по $mod\ n$. Для кількох значень множення виконується надзвичайно повільно, і противник знає, для яких саме. Тому якщо час виконання алгоритму дешифрування завжди помітно збільшується, коли ця конкретна ітерація виконується при значенні біта, що дорівнює 1, то, виявивши відповідну затримку, можна зробити висновок про те, що даний біт дорівнює 1. Якщо ж для цілої серії спостережень алгоритм завжди виконується достатньо швидко, то цей біт повинен бути рівним 0.

На практиці реалізації зведення в ступінь в класах відрахувань не мають настільки різко виражених варіацій в часі, щоб тривалість виконання однієї ітерації могла перевищити середній час виконання всього алгоритму в цілому. Але мають місце в реальності варіації все ж достатні для того, щоб їх аналіз виявився практично корисним.

Хоча аналіз витрат часу і виявляється серйозною загрозою, проти нього використовуються прості контрзаходи, наприклад, такі:

1. Постійний час виконання операції піднесення до степеня. Зміна алгоритму таким чином, щоб всі зведення в ступінь займали один і той ж час від початку виконання до повернення результату. Зробити це просто, але при цьому збільшується загальний час виконання алгоритму.

2. Випадкові затримки. Менший вплив на загальний час виконання викликає додавання в алгоритм піднесення до степеня випадкових затримок,

що зменшує користь від аналізу тимчасових витрат. Але при цьому якщо захищається, додасть недостатньо перешкод, противник буде все ще в стані за допомогою додаткових вимірів провести криптоаналіз і в умовах впливу випадкових затримок.

3. Маскування. Множення шифрованого тексту на випадкове число, перед тим як виконувати зведення в ступінь. Це не дозволить противнику дізнатися, які біти шифрованого тексту оброблялися в комп'ютері і, таким чином, не дасть йому можливості провести порозрядний аналіз, який є істотною частиною підходу, заснованого на аналізі тимчасових витрат.

2.5 Визначення актуальності шифру RSA

В 2001 році алгоритм RSA визнаний найбільш поширеною асиметричною криптосистемою (з відкритим ключем) і часто називається стандартом. Існування такого стандарту є правда важливим для подальшого розвитку електронної комерції та економіки взагалі. Єдина для всіх система відкритих ключів дозволяє здійснювати документообіг з цифровими підписами між користувачами по всій планеті, використовуючи абсолютно різні програми на різних платформах. Ця можливість є необхідною умовою для розвитку електронного документообігу та діловодства онлайн.

Розповсюдження системи RSA дійшло до того, що його враховували при створенні нових стандартів криптосистем. Також при розробці стандартів для цифрових підписів. Для прикладу, у 1997 році було розроблено ANSI X7.28, що підтримує стандарт Digital Signature. Через три роки було введено ANSI X8.32, зорієнтований на цифрові підписи за алгоритмом RSA, що відповідає актуальній ситуації, зокрема, для фінансових та юридичних установ.

Недоліки безпечної, захищеної аутентифікації були основною перешкодою для переходу паперових документів в електронний вигляд.

Майже всі договори, контракти, офіційні папери установ, юридичні документи, інформація про клієнтів досі існують на папері. Саме юридичні обумовленості аутентифікації – фактор, що не дозволяв повного переходу на електронні угоди. Цифровий підпис, запропонований структурами RSA, є інструментом, який дозволить перетворити важливі друковані потоки документів у цифровий формат. Завдяки електронним цифровим підписам багато документів - паспорти, угоди, купівлі, оренди і так далі - тепер можуть існувати в електронній формі, а будь-яка паперова версія буде потрібна хіба що як копія електронного оригіналу, який буде доступний завжди через програмні засоби. Все це стало можливим із-за створення єдиного стандарту цифрового підпису на базі методу шифрування RSA.

Незважаючи на відносно невисокий показник криптонадійності при тій же довжині ключів, RSA-шифрування застосовується в багатьох галузях. Особливо поширено використовують при великому кругообігу електронної документації. RSA-шифрування використовується для захисту документів на середньому рівні відповідальності і легко знайде застосування в середньому та малому бізнесі, компаніях, що зберігають дані, які не потребують найвищих стандартів захисту.

2.6 Модель алгоритму аутентифікації

Аутентифікація - перевірка приналежності клієнту або сервера пред'явленого їм ідентифікатора - можна реалізувати на основі ЕЦП і відповідних сертифікатів відкритих ключів.

Аутентифікація можлива декількома способами:

1. Сервер посилає клієнту запит на підтвердження справжності зашифрований відкритим ключем клієнта, отриманим з сертифіката відкритого ключа клієнта. Клієнт розшифровує запит особистим закритим

ключем і повертає серверу, підтвердивши таким чином, що він є власником відповідного закритого ключа, і, отже, ідентифікаційні дані в сертифікаті належать саме йому.

2. Сервер надсилає запит на підтвердження справжності відкритим текстом. Клієнт відповідає на запит, підписавши його власної електронним цифровим підписом. Сервер перевіряє ЕЦП клієнта за допомогою відкритого ключа отриманого з сертифіката відкритого ключа клієнта і засвідчується в тому, що клієнт дійсно має відповідний особистий закритий ключ.

Описана схема називається протоколом доказу володіння, оскільки відправник доводить, що він володіє необхідним для дешифрування і створення електронного цифрового підпису особистим секретним ключем. Обидва запропонованих алгоритми можуть бути застосовані для цілей визначення факту підписання документу конкретним користувачем. Для цього в системі достатньо ввести дані його аккаунту, а система, непомітно для користувача, проведе розглянуті операції та видасть результат.

Алгоритм перевірки електронного підпису наступний. На першому етапі одержувач повідомлення створює свою власну версію хеш-функції підписаного документа. На другому етапі хеш-функція розшифровується в повідомленні за допомогою відкритого ключа відправника. На третьому етапі проводиться порівняння двох хешей. Їх збіг гарантує одночасно достовірність змісту документа і авторство відправника.

Електронний цифровий підпис, як і будь-які інші дані, може передаватися разом з підписаними, захищеними самим відправником даними. Шифрування та електронний підпис можна успішно використовувати разом. Ви можете спочатку підписати документ приватним закритим ключем, а потім зашифрувати його відкритим ключем адресата. Підпис ідентифікує особу, шифрування захищає лист від сторонніх очей.

2.7 Застосування хеш-функцій в контексті ЕЦП

Документи, що підписуються мають різну, часто велику, довжину. Саме тому в випадку застосування з ЕЦП зручно використовувати документ не в початковому вигляді, а тільки його хеш. Хеш – це результат роботи хеш-функцій або функцій згортання. Обрахування хеша використовує саме криптографічні хеш-функції для забезпечення виявлення змін документів були під час перевірки підпису. Хеш-функції не є частиною алгоритму ЕЦП. Тому може бути використана будь-яка або не використовуватись взагалі.

Хеш-функція призначена для стиснення послідовності вхідних даних довільної довжини в бітовий рядок попередньо визначеного розміру. Це зазвичай кілька десятків або сотень біт. Аналіз, який використовує хеш-функцію, часто використовується для контролю цілісності критичних системних файлів, важливих програм, важливих даних. Моніторинг можна проводити за необхідності та на регулярній основі.

Спочатку потрібно визначити, цілісність яких файлів необхідно відстежувати. Для кожного файлу його значення обчислюється за допомогою спеціального алгоритму, який зберігає результат. Через деякий час проводиться обрахунок значення його хеша з збереженням результату. Якщо значення відрізняються, інформація, що міститься у файлі, була змінена.

Хеш-функції повинні володіти наступними характеристиками:

- повинні мати можливість виконувати перетворення даних довільної довжини у дані фіксованої довжини;

- повинні мати відкритий алгоритм, щоб мати можливість вивчити його криптографічну стійкість;

- вона повинна бути односторонньою, тобто не повинно бути математичного способу визначення вихідних даних з результату;

- ймовірність зіткнень, тобто ймовірність того, що значення хеш-функцій двох різних документів (незалежно від їх довжини) будуть рівні, має бути незначною.

- вона не повинна вимагати великих обчислювальних ресурсів;
- найменша зміна вхідних даних повинна істотно змінити результат.

Використання хеш-функції, в алгоритмі створення ЕЦП забезпечують деякі переваги, зокрема:

- складність обрахунків;
- сумісність;
- цілісність.

За рахунок того, що хеш-функція значно зменшує обсяг документу, обрахунки на виході стають швидшими. Тому створювати хеш документу і тільки потім підписувати його – це оптимальне рішення в рамках поставленої задачі. Хеш можна використовувати для перетворення будь-якого вхідного тексту у потрібний формат. Якщо не використовувати функції стиснення, то документи великого розміру інколи розділяти на менші блоки задля застосування ЕЦП. Загалом, односпрямованість не розуміється як неможливість, а скоріше як велика складність при поверненні повідомлення від свого хешу, оскільки в даний час не існує хеш-функції з доведеною односпрямованістю.

3 ОПИС ВИКОРИСТАНИХ ПРОГРАМНИХ ЗАСОБІВ

Аналізуючи поставлену задачу, вирішено розроблювати програмне забезпечення у вигляді веб-додатку. Це оптимальне рішення, тому що мета роботи передбачає застосування програми користувачами, які знаходяться віддалено один від одного. Клієнт веб-застосунків не операційна система, а встановлений браузер, що дозволяє зробити систему універсальнішою та завідома сумісною з більшою кількістю пристроїв.

3.1 Засоби розробки

Логіка додатку та внутрішні обчислювальні методи реалізовані за допомою мови програмування JAVA 8 версії.

Середовищем розробки обрано IntelliJ IDEA.

Для створення користувацького інтерфейсу використано мову розмітки HTML та мову стилів CSS.

Для збірки проекту використано фреймворк Maven.

Для зручності написання та побудови структури проекту використано фреймворк Spring.

Використані зовнішні бібліотеки, наприклад BitInteger для множення великих чисел без колізії.

В якості сховища даних обрано файл json формату, дані в якому видозмінені за допомогою методів сералізації/десеріалізації.

3.1.1 Мова програмування Java

Java є мовою програмування високого рівня та обчислювальною платформою, розробленою компанією Sun Microsystems у 1995 році. З тих пір мова регулярно поповнюється новими версіями.

Виходячи з переваг Java, вона отримала широку популярність. Побудовано декілька конфігурацій для різних типів платформ, включаючи Java SE для Macintosh, Windows і UNIX, Java ME для мобільних додатків і Java EE для корпоративних додатків.

Завдяки зростаючому значенню веб-додатків і мобільних додатків, Java сьогодні є основою для більшості мережевих програм і вважається корисною для серверної розробки, роботою з веб-вмістом, створення корпоративного програмного забезпечення, ігор і мобільних додатків.

Переваги Java:

- Java пропонує більш високу крос-функціональність і портативність, оскільки

програми, написані на одній платформі, можуть працювати на настільних комп'ютерах, мобільних телефонах, вбудованих системах;

- проста, об'єктно-орієнтована, розподілена;

- підтримує багатопоточність;

- Java є безкоштовною, і пропонує мультимедійну та мережеву підтримку;

- Java - це зріла мова, тому вона більш стабільна і передбачувана. Бібліотека класів Java дає можливість розробки на різних платформах;

- Дуже популярна на корпоративному, вбудованому та мережевому рівнях, Java має велику активну спільноту користувачів і доступну підтримку;

- Підвищення мовної різноманітності, про що свідчить сумісність Java з Scala, Groovy, JRuby і Clojure;

- Відносно бездоганна сумісність з однієї версії до іншої

В кожній версії Java є свої особливості. Для кодування системи було обрано версію 8 цієї мови, оскільки її можна вважати найбільш стабільною та поширеною, без експериментальних доповнень, які не пройшли перевірку часом та достатньою кількістю реальних проектів.

3.1.2 Фреймворк Maven

Maven - це інструмент управління проектами, який надає розробникам готову структуру життєвого циклу. Розробник може автоматизувати побудову інфраструктури проекту в короткий час, оскільки Maven використовує стандартну розкладку каталогів і побудову життєвого циклу за замовчуванням. Maven може налаштувати спосіб роботи відповідно до стандартів за дуже короткий час. Оскільки більшість установок проекту є простими і довговічними, Maven спрощує процес розробки під час створення звітів, перевірок, побудови структури проекту та автоматизованого тестування. Maven надає розробникам ряд переваг:

- управління та впровадження повного циклу проекту. Компіляції коду, тестування модулів, генерація збірки (файли jar та war);

- легке керування зовнішніми залежностями проекту на основі Java і зберігання проекту Java невеликим.

- багаторівневе управління проектами, тобто обмін функціональними можливостями одного проекту з іншими численними проектами з використанням управління залежностями.

- Краще звітування про помилки та цілісність - Maven покращила звітність про помилки, і вона надає вам посилання на вікі-сторінку Maven, де ви отримаєте повний опис помилки.

Підводячи підсумок, Maven спрощує і стандартизує процес створення проекту. Він обробляє компіляцію, розповсюдження, документацію, командну співпрацю та інші завдання. Maven підвищує можливості повторного

використання і піклується про більшість завдань, пов'язаних з будовою структури проекту.

3.1.3 Фреймворк Spring

Spring - найпопулярніша платформа розробки додатків для Java. Мільйони розробників у всьому світі використовують Spring Framework для створення високопродуктивного та повторно використовуваного коду, що легко тестується.

Spring framework - відкрита платформа Java. Він був написаний Родом Джонсоном і вперше був випущений під ліцензією Apache 2.0 у червні 2003 року. Основні особливості Spring Framework можна використовувати при розробці будь-якого Java-додатка, для створення веб-додатків поверх платформи Java EE. Цільові функції Spring сприяють хорошій практиці програмування.

Переваги використання Spring Framework:

- Spring дозволяє розробникам розробляти програми з використанням POJO. Перевагою використання тільки POJO є те, що вам не потрібен контейнер EJB, такий як сервер додатків;

- Spring організований модульно. Незважаючи на те, що кількість пакетів і класів є істотною, вам потрібен лише нешироκий їх спектр для кожної задачі.

- Spring не винаходить колесо, замість цього він дійсно використовує деякі з існуючих технологій, таких як кілька ORM фреймворків, JEE, Quartz і JDK таймери, а також інші технології перегляду та відображення.

Веб-фреймворк Spring - це добре продуманий веб-фреймворк, базований на MVC, який надає чудову альтернативу веб-фреймворкам, таким як Struts або інші надмірно складні або менш популярні веб-фреймворки.

Технологія, з якою найбільш ототожнюється Spring, - це Dependency Injection (ін'єкції залежності) від інверсії контролю. Інверсія контролю є

загальною концепцією, і вона може бути виражена багатьма різними способами.

ін'єкція залежностей може відбуватися шляхом передачі параметрів конструктору або POST запиту з використанням методів сеттера.

3.1.4 Thymeleaf шаблонізатор

Thymeleaf - бібліотека Java. Це шаблонний движок XML/XHTML/HTML5, здатний застосувати набір перетворень до шаблонів для відображення даних та тексту, створених вашими програмами. Він краще підходить для обслуговування XHTML / HTML5 у веб-додатках, але він може обробляти будь-який файл XML, як в Інтернеті так і в окремих програмах.

Основна мета Thymeleaf - забезпечити елегантний і добре сформований спосіб створення шаблонів. Щоб досягти цього, він базується на тегах XML і атрибутах, які визначають виконання попередньо визначеної логіки на DOM (Document Object Model), замість того, щоб явно писати цю логіку як код всередині шаблону.

Його архітектура дозволяє здійснювати швидку обробку шаблонів, спираючись на інтелектуальне кешування проаналізованих файлів, щоб використовувати найменш можливу кількість операцій введення / виводу під час виконання. Важливо, що Thymeleaf було розроблено з самого початку з урахуванням XML і веб-стандартів, що дозволяє створювати шаблони, які легко перевіряються та редагуються, якщо це необхідно для вас.

Головною метою Thymeleaf є створення елементарних природних шаблонів у вашому робочому процесі розробки - HTML, який можна правильно відобразити в браузерах, а також стати статичними прототипами, що дозволяє інтегрувати структурні елементи програми в новий вигляд.

Eclipse, IntelliJ IDEA, Spring, Play, навіть найсучасніші API для моделей Model View Controller nf Java EE 8. Thymeleaf підтримує всі популярні інструменти. Thymeleaf ідеально підходить для сучасної веб-розробки HTML5

в рамках Java проектів. Шаблони HTML, написані в Thymeleaf, виглядають і працюють як HTML, дозволяючи реальним шаблонам, які виконуються у вашому додатку, працювати як корисні артефакти дизайну.

3.1.5 Розмітка HTML 5 та стилі CSS 3

HTML (HyperText Markup Language) - мова розмітки гіпертексту, призначена для створення Web-сторінок. Під гіпертекстом в цьому випадку розуміється текст, пов'язаний з іншими текстами покажчиками-посиланнями.

HTML являє собою досить простий набір кодів, які описують структуру документа. HTML дозволяє виділити в тексті окремі логічні частини (заголовки, абзаци, списки і т.д.), помістити на Web-сторінку підготовлену фотографію або картинку, організувати на сторінці посилання для зв'язку з іншими документами. Будь-який HTML документ складається з набору елементів, початок і кінець кожного елемента позначається спеціальними позначками - тегами. Між тегами ви побачите уміст елемента - дані або текст (наприклад елемент заголовка: <title> Hello, world! </ Title>, де <title> - відкриває тег, </ title> - закриває). Крім даних, елемент може мати атрибути, що визначають властивості елемента (наприклад спосіб вирівнювання тексту, колір шрифту, розмір картинки).

Мова HTML5 робить акцент на спрощення розмітки, необхідної для створення відповідних стандартів сторінок і об'єднання всього необхідного з CSS і Java кодом, а також файлів зображень.

HTML можна назвати основною мовою Всесвітньої павутини. Більшість веб-сторінок, розміщених в Інтернеті, написані в якійсь з варіацій HTML. За допомогою нього розробники визначають те, як мультимедіа, текст або гіперпосилання відображатимуться серед іншого контенту в браузері. Починаючи від елементів, які встановлюють зв'язку з вашим документом до елементів які роблять ці документи інтерактивними (наприклад форми) - все це є складовими частинами HTML.

CSS - це фактично мова стилів, який визначає відображення HTML-документів. CSS працює зі шрифтами, з символами і фоном, полями, рядками, з висотою і з шириною елементів відображення, з фоновими зображеннями, з позиціонуванням елементів і тому подібне. Якщо HTML необхідний для структурування змісту сторінки, то CSS необхідний для того, щоб форматувати цю структуру.

Використання CSS полегшує створення якісних сайтів, дозволяючи задати стилі окремих елементів сторінок сайту в особливих css-файлах, щоб в подальшому бути впевненим в тому, що всі сторінки сайту будуть витримані в єдиному стилі.

Для форматування сторінок користувацького інтерфейсу використана 3 версія мови CSS, яка має свої переваги:

- використання таблиці стилів. Задовго до розробки концепції каскадних таблиць стилів CSS3 використовувалася HTML-розмітка. Але з введенням каскадних таблиць все це стало можливим задавати в окремій таблиці стилів, в результаті чого користувачі отримали простий та зручний інструмент. З цим пов'язаний ще один плюс CSS3 - стало простіше вносити зміни: можна змінювати окремі модулі, які інтегруються із загальною системою.

- диференціація і ізоляція. Диференціація забезпечує більш ефективну і зручну форму. Модульний підхід допомагає розвивати і підтримувати системи, даючи велику гнучкість.

- макет декількох колонок. Модуль на основі декількох колонок (Multi-Column Module) - важлива функція CSS3, що дозволяє помістити текст в декілька стовпців.

- гнучкість у використанні. Концепція каскадних таблиць стилів дозволяє приєднати інформацію CSS-стилю у вигляді окремого документа або у вигляді вкладення усередині HTML-документа. Також можна імпортувати кілька таблиць стилів в будь-якому місці.

3.2 Середовище розробки

Без сумніву, найпопулярнішим середовищем для написання коду на мові програмування Java є IntelliJ IDEA від Jet Brains.

Використано новітнє та поширене середовище, щоб уникнути часу на вивчення принципів роботи з технологіями, які рідко використовуються для розробки подібних проектів, але отримати доступ до достатньо великого та ефективного набору засобів редагування та налаштування Java-проектів.

IntelliJ IDEA - це, перш за все, середовище розробки для Java, включаючи Java 8. З цією мовою вона дружить найбільше, відмінно його розуміє і допомагає в написанні розробнику. Але це не означає, що все закінчується на Java і власною мовою Kotlin. Не менш важливими є їх розробки таких передових технологій, як Groovy, Scala і інших. Вони поєднують в собі можливості динозаврів, як Java разом з функціоналом Ruby, Smalltalk і інших. Однією з сильних сторін в JetBrains вважають підтримку широкого кола технологій.

IntelliJ IDEA являє собою високотехнологічний комплекс тісно інтегрованих інструментів програмування, що включає інтелектуальний редактор вихідних текстів з розвиненими засобами автоматизації, потужні інструменти рефакторинга коду, вбудовану підтримку технологій J2EE, механізми інтеграції з середовищем тестування Ant / JUnit і системами управління версіями, унікальний інструмент оптимізації та перевірки коду Code Inspection, а також інноваційний візуальний конструктор графічних інтерфейсів.

Висновки до розділу

У даному розділі розглянуто інструменти реалізації програмного продукту, а також описані додаткові модулі програми, що використовуються для забезпечення виконання поставлених задач.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Система була спроектована відповідно до основних вимог компонентного програмування. Кожний модуль системи відповідає за реалізацію чіткого та мінімального переліку функцій та взаємодії з іншими компонентами за допомогою гнучкого інтерфейсу.

Для роботи web-інтерфейсу необхідно мати сучасний web-браузер та стабільний доступ до мережі Інтернет.

Для запуску системи на сервері необхідно принаймні 2 гігабайти оперативної пам’яті та Java Development Kit версії 8 або вище.

Шар представлення, реалізований у вигляді web-інтерфейсу, сполучається контролером із шаром сервісу, в якому можна відокремити самостійні елементи, відповідальні за збереження інформації про користувачів та безпосередньо створення і перевірку цифрового підпису.

Принципову архітектуру системи зображено на рисунку 4.1.

Рисунок 4.1 — Загальна архітектура створеної системи

Деякі модулі, наприклад опис структури даних про користувача, використовуються одночасно у роботі більш ніж одного іншого модуля. Можливість повторного використання характеризує високу відповідність системи стандартам об’єктно-орієнтованого програмування.

4.1 Опис функціональності системи

Головний актор системи може виконувати дії двох типів:

— управління персональним кабінетом;

— роботи з можливостями електронного підпису.

Діаграма прецедентів, що зображена на рисунку 4.2, описує набір дій, які створюють цілісний та повний функціональний сценарій роботи користувача зі системою.

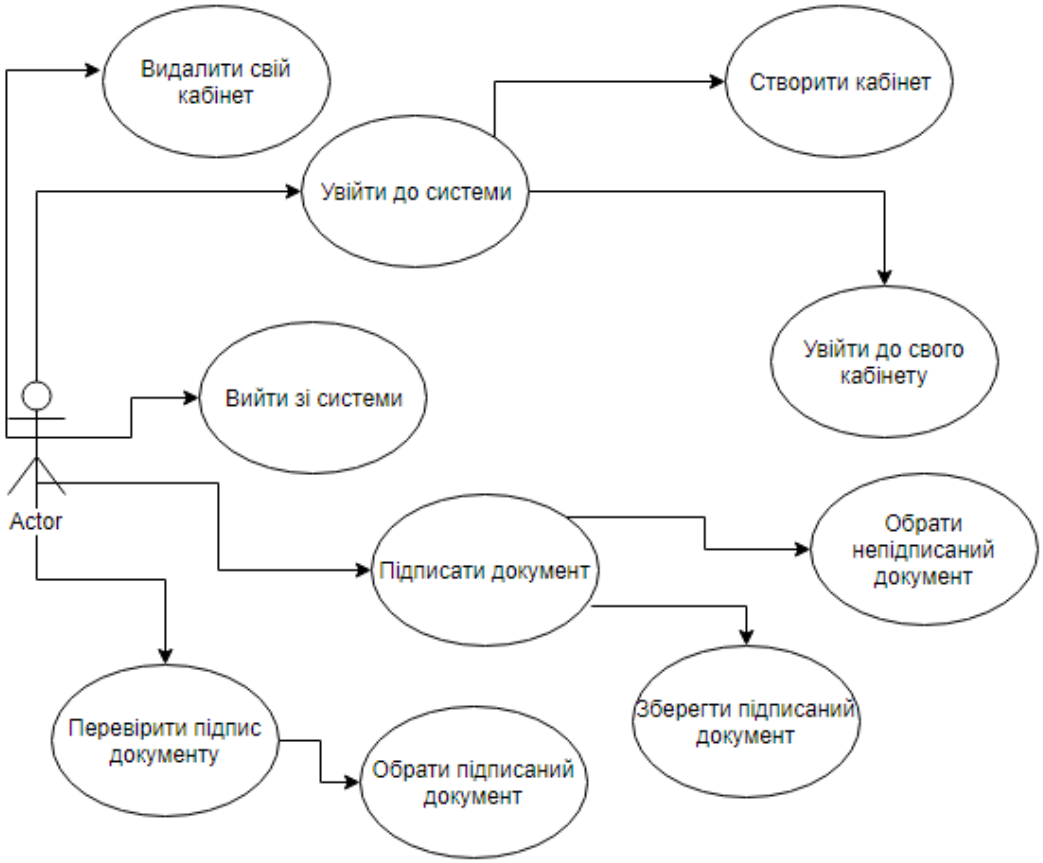

Рисунок 4.2 — Діаграма прецедентів системи

Виконання певних дій передбачає взаємодію з файловою системою користувача.

4.2 Структура бази даних системи

Оскільки система працює з конфіденційними даними, вони повинні бути збережені у захищеному вигляді з метою запобігання можливим спробам зловмисників їх використати для зменшення надійності роботи системи або отримання доступу до персональної інформації користувачів.

Обсяг даних, що потребують довгострокового збереження, є достатньо малим та стосується лише інформації, пов’язаної з персональними кабінетами користувачів.

Таким чином, база даних містить інформацію про

— імена (логіни) та паролі;

— RSA ключі наявних користувачів.

4.3 Структура бізнес-моделі

Головною складовою системи, де реалізовані всі сервіси, які вона може надавати користувачам, є її бізнес-модель.

Дана частина системи реалізує функції:

— підпису тексту;

— перевірки правильності підпису для конкретного тексту;

— зчитування даних про користувачів зі сховища даних;

— оновлення даних про користувачів у сховищі даних.

Дані про користувача зберігаються у класі, що містить такі поля даних:

— ім’я користувача;

— пароль персонального кабінету;

— пара унікальних RSA ключів;

— об’єкт цифрового підпису, що отримує початкові значення своїх параметрів при створенні кабінету.

Задачею модулю взаємодії зі сховищем даних є взаємна конвертація між вище описаною структурою даних та фізичним форматом, у якому вони зберігаються у певних файлах.

Модуль електронного підпису також взаємодіє з даною структурою, коли використовує ключі шифрування конкретного користувача з метою створення чи перевірки електронного підпису.

4.4 Розробка інтерфейсу взаємодії з користувачем

Користувач взаємодіє з системою через її web-інтерфейс. Шар представлення було оформлено у вигляді шаблонів, що наповнюються інформацією через контролер, конкретні методи якого викликаються, коли користувач активує певні елементи графічного інтерфейсу.

Інформація від користувача оброблюється методами сервісу та повертається або замінюється новою. Контролер розроблений з точки зору незалежності від реалізації кінцевого інтерфейсу. Це дозволяє легко переходити від web-застосунку до desktop-програми і навпаки, оскільки такий перехід не потребує суттєвих зміни шару сервісу та навіть великої кількості модифікацій безпосередньо у головному контролері.

4.5 Інструкція з використання програмного продукту

Домашньою сторінкою застосунку містить форму логіну до системи, яка зображена на рисунку 5.1.

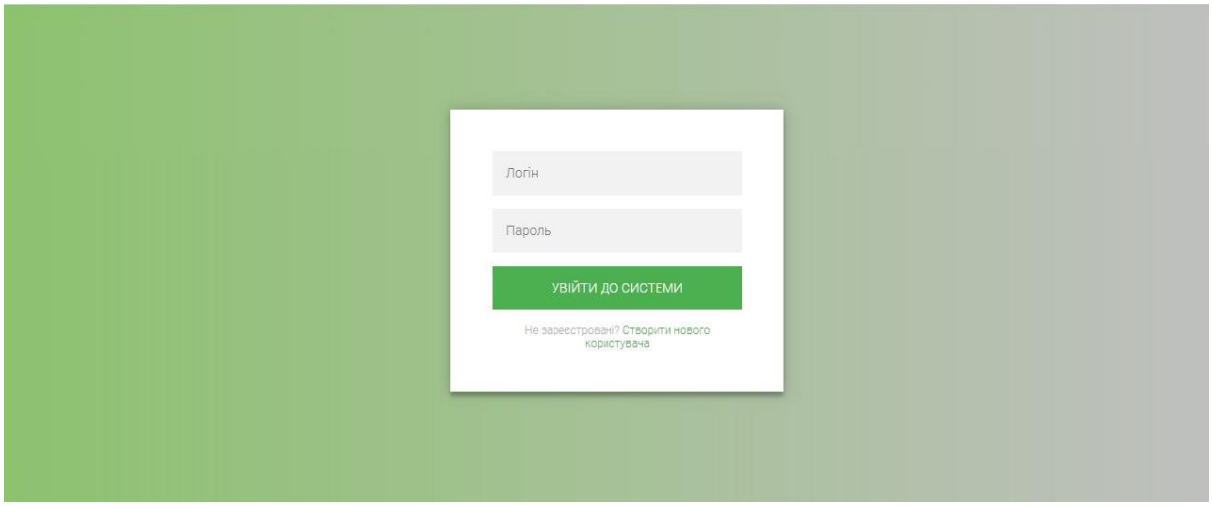

Рисунок 5.1 — Форма логіну

Якщо користувач ще не має свого персонального кабінету, він має можливість створення нового. Для цього передбачено форму реєстрації, яка зображена на рисунку 5.2.

Рисунок 5.2 — Форма реєстрації

Успішна реєстрація або логін дозволяють здійснити перехід до основної сторінки, де користувач може виконати дії, пов’язані з цифровим підписом документів (рисунок 5.3).

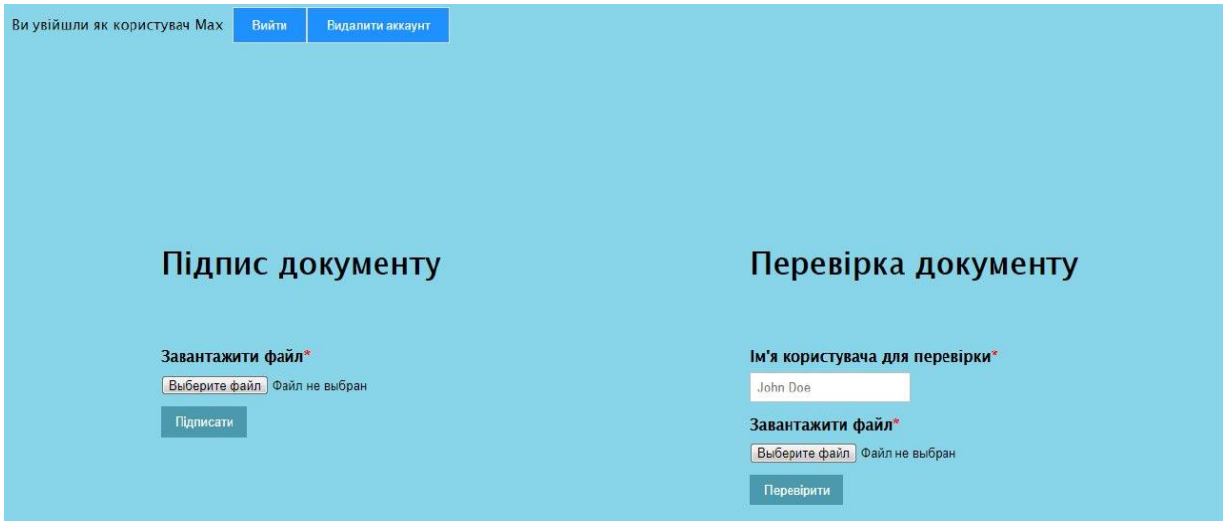

Рисунок 5.3 — Основна сторінка web-інтерфейсу

Користувач може обрати документ і шлях для збереження результату та згенерувати цифровий підпис для даного документу (рисунок 5.4).

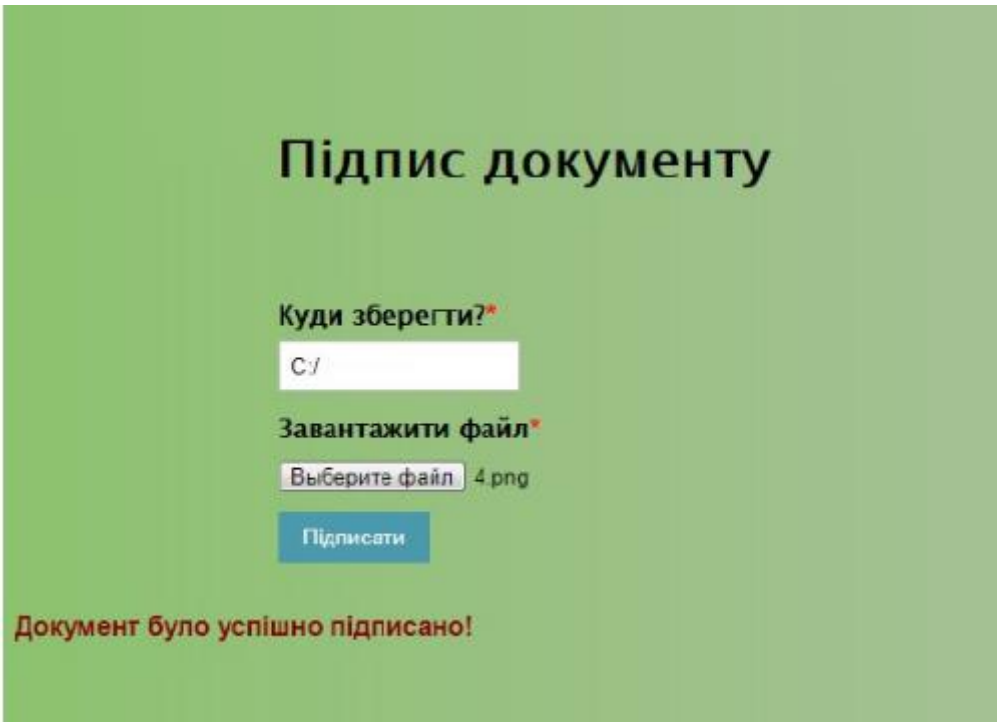

Рисунок 5.4 — Результат прикріплення цифрового підпису до документу користувача

Подібним чином виглядає перевірка підписаного документу (рисунок 5.5). Необхідно задати ім’я користувача, відносно якого необхідно провести перевірку, та надати підписаний документ з локальної файлової системи.

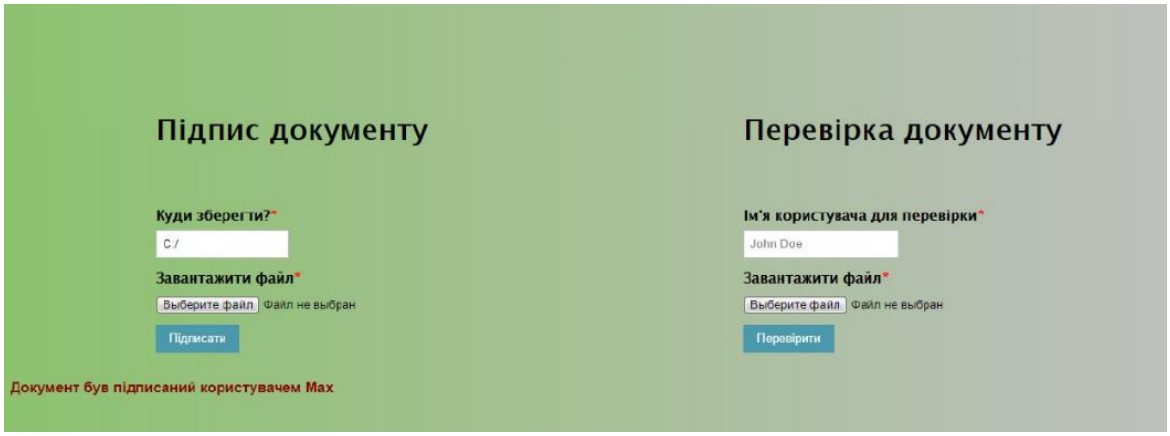

Рисунок 5.5 — Результат перевірки документу на цифровий підпис конкретним користувачем

У разі потреби користувач може видалити свій персональний кабінет (рисунок 5.6).

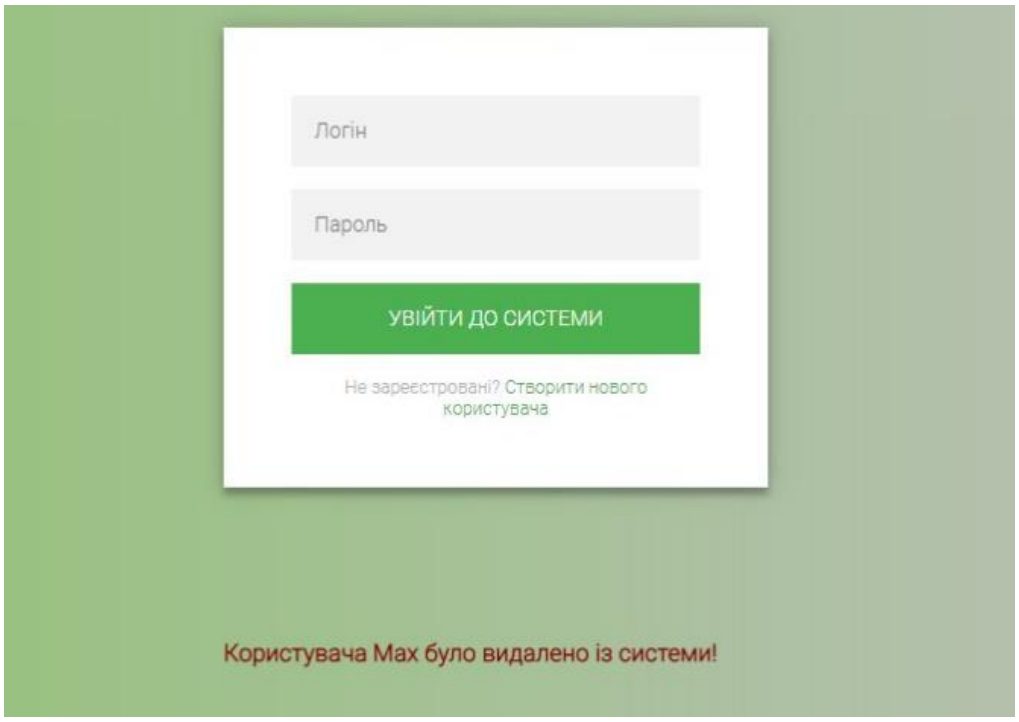

Рисунок 5.6 — Результат видалення користувачем свого персонального кабінету з системи

Як видно з даного рисунку, після видалення відбувається миттєве посилання на сторінку авторизації з відображенням діагностичного повідомлення про успішне завершення попередньої операції.

ВИСНОВКИ

У ході дипломної роботи досліджено механізм цифрового підпису та існуючі алгоритми, які надають можливість як підписувати документи, так і перевіряти, чи були ці документи вже підписані певними користувачами. Проаналізовано їх слабкі місця.

Розроблений програмний продукт дозволяє ефективно автоматизувати процедуру затвердження та захисту електронних документів.

Обрано якісні засоби розробки та надійну архітектуру, щоб система могла працювати з великою кількістю користувачів та при цьому зберігати криптографічну стійкість та безперебійність роботи.

Робота системи була перевірена на тестових випадках, які моделюють прецеденти сценарію роботи користувача із системою.

Розроблений застосунок може бути використаний різними користувачами. Показником для впровадження системи у використання є необхідність захисту документації та її обігу у професійному, науковому, виробничому середовищі.

Розроблена система показала високу ступінь ефективності та має широкий спектр потенційного використання.

У подальшому застосунок може бути покращений за рахунок вдосконалення алгоритмів шифрування та додавання нових функцій, в тому числі управління персональним кабінетом. Можливою є інтеграція системи з поштовими клієнтами для автоматизації процесу передачі підписаних документів через мережу Інтернет.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Саломаа А. Криптография с открытым ключом. Пер. с англ. – М.: Мир, 1995. – 318 с.
2. Смарт Н. Криптография. Москва: Техносфера, 2005. – 528 с.
3. Бернет С., Пейн С. Криптография. Официальное руководство RSA Security. - М.: Бином-Пресс, 2002. - 392 с.
4. Anthony Gore — Full-Stack Vue.js 2 and Laravel 5 [Електронний ресурс]. — 2015.
5. Соловяненко Н.И. Юридическая роль электронной подписи в электронной коммерции. - М.: МЗ Пресс, 2002. - 215 с.
6. Халиков Р. Субъекты электронного документа в банковской сфере // Информационное право. - 2006. - 121 с.
7. Дэвид Флэнэген Java in a Nutshell .- O'Reilly & Associates, Inc., 1997, Издательская группа BHV, Киев, 1998. – 124 с.
8. Язык программирования Java SE 8. Подробное описание. - М.: Вильямс, 2015. - 672 с.
9. Семенов Ю.А. Протоколы Internet // М.: Проспект, 2011. – 114 с.
10. Беляев А.В. Методы и средства защиты информации // ЧФ СПбГТУ, 2010. – 142с
11. Венбо М. Современная криптография. Теория и практика // М.: Вильямс, 2005. — 768 с.
12. Петров, А. А. Компьютерная безопасность. Криптографические методы защиты / А. А. Петров. – М. : ДМК, 2000. – 448 с.
13. Шаньгин, В. Ф. Информационная безопасность компьютерных систем и сетей : учеб. пособие / В. Ф. Шаньгин. – М. : ИД «Форум»: Инфра-М, 2011. – 416 с.
14. Торокин, А. А. Инженерно-техническая защита информации / А. А. Торокин. – М. : Гелиос АРВ, 2005. – 360 с.

15. Информационная технология. Криптографическая защита информации. Функция хеширования. — Введ. 2012–07–08. — М.: Стандартинформ, 2012. — 38 с.
16. Джошуа Б. Java. Эффективное программирование / Блох Джошуа. - М.: ЛОРИ, 2014. - 292 с.
17. Карабин, Петр Язык программирования Java: Создание интерактивных приложений для Internet / Петр Карабин. - М.: Познавательная книга плюс, 2010. - 224 с.
18. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты // М.: Триумф, 2002. — 816 с.Бузов, Геннадий Алексеевич Защита информации ограниченного доступа от утечки по техническим каналам / Бузов Геннадий Алексеевич. - М.: Горячая линия - Телеком, 2016. - 186 с.
19. Хорстманн, Кей С. Java SE 8. Вводный курс / Хорстманн Кей С.. - М.: Диалектика / Вильямс, 2014. - 898 с.
20. Язык программирования Java SE 8. Подробное описание. - М.: Вильямс, 2015. - 672 с.
21. Хуанг, Т.С.ред. Быстрые алгоритмы в цифровой обработке изображений. Преобразования и медианные фильтры / Т.С.ред. Хуанг. - М., 2017. - 432 с.
22. Юрасов, А.В. Основы электронной коммерции / А.В. Юрасов. - М.: Горячая линия - Телеком, 2017. - 480 с.
23. Тимошин П. Перспективы развития и использования систем электронной цифровой подписи. Флинта. 2014 г. 28 с.
24. Гудман, Д. Java Script и DHTML. Сборник для профессионалов / Д. Гудман. - М.: СПб: Питер, 2004. - 523 с.
25. Ноултон, П. Java 2 в подлиннике / П. Ноултон, Г. Шилдт. - М.: СПб: BHV, 2001. - 451 с.

ДОДАТОК А

Система захисту даних на базі методу шифрування RSA.

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51168_19Б

Аркушів 1

Київ 2018

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51168_19Б	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51168_19Б 12-1	SigningController.java CustomRSA.java Signing.java	Основні компоненти
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТВ51168_19Б 13-1	Додаток В.doc	Опис програмного модуля

ДОДАТОК Б

Система захисту даних на базі методу шифрування RSA.

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51168_19Б 12-1

Аркушів 7

Київ 2018

```
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Controller
@RequestMapping("/")
public class SigningController {
    private Model model;

    public SigningController(){
        this.model = new Model();
    }

    @RequestMapping("")
    public String home(Map<String, Object> model){
        return "authorization";
    }

    @RequestMapping(value = "", params = {"form"})
    public String homeFormSpecific(@RequestParam String form,
                                   Map<String, Object> model){
        if(form.equals("create")){
            model.put("showCreate", true);
        }
        else{
            model.put("showCreate", false);
        }
        return "authorization";
    }

    @RequestMapping(value = "login", method= RequestMethod.POST)
    public String login(@RequestParam String username,
                        @RequestParam String password,
                        Map<String, Object> viewModel){
        User user = model.findUser(username, password);
        if(user != null){
            model.setCurrentUser(user);
            viewModel.put("currentUsername", username);
```

```
                return "activity";
            }
            else{
                viewModel.put("message", "Користувача з такими даними не знайдено!
" +
                        "Спробуйте створити нового користувача");
                return "authorization";
            }
        }

        @RequestMapping(value = "create", method= RequestMethod.POST)
        public String createUser(@RequestParam String username,
                                 @RequestParam String password,
                                Map<String, Object> viewModel){

        @RequestMapping(value = "delete", method= RequestMethod.POST)
        public String removeUser(Map<String, Object> viewModel){
            model.removeUser(model.getCurrentUser());
            viewModel.put("message", "Користувача "
                    + model.getCurrentUser().getUsername() + " було видалено із
системи!");
            return "authorization";
        }


        @RequestMapping(value = "verify", method= RequestMethod.POST)
        public String verifySignature(@ModelAttribute("object") Object filebject,
                                      @RequestParam String username,
                                      @RequestParam("file") MultipartFile file,
BindingResult bindingResult,
                                      Map<String, Object> viewModel){
            if(!bindingResult.hasErrors()) {

                byte[] buffer = new byte[]{0};
                try {
                    buffer = file.getBytes();
                }
                catch(IOException ex){
                    System.err.println(ex.getMessage());
                    viewModel.put("message", "Документ не був підписаний
користувачем " + username);
```

```
                }
                String text = Model.EMPTY_FILE_MARKER;
                try (InputStream in = new ByteArrayInputStream(buffer)) {
                    text = new BufferedReader(new InputStreamReader(in))
                            .lines().collect(Collectors.joining("\n"));
                } catch (IOException ex) {
                    System.err.println(ex.getMessage());
                    viewModel.put("message",    "Документ    не    був    підписаний
користувачем " + username);
                }

                try {
                    String[] signatures = ((String)

text.subSequence(text.indexOf(Model.DIGITAL_SIGNATURE_STARTING_MARKER) +

Model.DIGITAL_SIGNATURE_STARTING_MARKER.length()
                                    ,
text.length())).split(Model.DIGITAL_SIGNATURE_CONTINUATION_MARKER);
                    String     fileText     =     (String)     text.subSequence(0,
text.indexOf(Model.DIGITAL_SIGNATURE_STARTING_MARKER));
                    boolean success = false;
                    for(String signature : signatures){
                        if
(Signing.verifySignature(model.findUserUnsafely(username), signature, fileText)) {
                            viewModel.put("message",   "Документ   був   підписаний
користувачем " + username);
                            success = true;
                        }
                    }

                    if(!success) {
                        viewModel.put("message",   "Документ   не   був   підписаний
користувачем " + username);
                    }
                }
                catch(Exception e){
                    viewModel.put("message",    "Документ    не    був    підписаний
користувачем " + username);
                }
            }
```

```
            else {
                viewModel.put("message", "Документ не був підписаний користувачем "
+ username);
            }
            viewModel.put("currentUsername",
model.getCurrentUser().getUsername());
            return "activity";

        }

        @RequestMapping(value = "sign", method= RequestMethod.POST)
        public String putSignature(@ModelAttribute("object") Object fileObject,
                                   @RequestParam String path,
            @RequestParam("file")     MultipartFile     file,     BindingResult
bindingResult,
                                   Map<String, Object> viewModel){

            if(!bindingResult.hasErrors()) {

                byte[] buffer = new byte[]{0};
                try {
                    buffer = file.getBytes();
                }
                catch(IOException ex){
                    System.err.println(ex.getMessage());
                    viewModel.put("message", "Підписати документ не вдалося :(");
                }
                String text = Model.EMPTY_FILE_MARKER;
                try (InputStream in = new ByteArrayInputStream(buffer)) {
                    text = new BufferedReader(new InputStreamReader(in))
                            .lines().collect(Collectors.joining("\n"));
                } catch (IOException ex) {
                    System.err.println(ex.getMessage());
                    viewModel.put("message", "Підписати документ не вдалося :(");
                }
                try  (FileWriter  fileWriter  =  new  FileWriter(new  File(path  +
file.getOriginalFilename()), false)) {
                    if(text.contains(Model.DIGITAL_SIGNATURE_STARTING_MARKER)){
                        String       originalText       =       text.substring(0,
text.indexOf(Model.DIGITAL_SIGNATURE_STARTING_MARKER));
```

```
                        fileWriter.write(text                                    +
Model.DIGITAL_SIGNATURE_CONTINUATION_MARKER
                                +   Signing.createSignature(model.getCurrentUser(),
originalText));
                    }
                    else {
                        fileWriter.write(text                                    +
Model.DIGITAL_SIGNATURE_STARTING_MARKER
                                +   Signing.createSignature(model.getCurrentUser(),
text));
                    }
                    viewModel.put("message", "Документ було успішно підписано!");
                } catch (IOException ex) {
                    System.err.println(ex.getMessage());
                    viewModel.put("message", "Підписати документ не вдалося :(");
                }
            }
            else{
                viewModel.put("message", "Підписати документ не вдалося :(");
            }
            viewModel.put("currentUsername",
model.getCurrentUser().getUsername());
            return "activity";
        }

        public String getCurrentUsername(){
            return model.getCurrentUser().getUsername();
        }
    }    public CustomRSA()
        {
            r = new Random();
            p = BigInteger.probablePrime(bitlength, r);
            q = BigInteger.probablePrime(bitlength, r);
            N = p.multiply(q);
            phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
            e = BigInteger.probablePrime(bitlength / 2, r);
            while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) <
0)
            {
                e.add(BigInteger.ONE);
            }
```

```
            d = e.modInverse(phi);
        }

        public CustomRSA(BigInteger e, BigInteger d, BigInteger N)
        {
            this.e = e;
            this.d = d;
            this.N = N;
        }

        //Тестовий метод для перевірки правильності роботи
        @SuppressWarnings("deprecation")
        public static void main(String[] args) throws IOException
        {
            CustomRSA rsa = new CustomRSA();
            DataInputStream in = new DataInputStream(System.in);
            String teststring;
            System.out.println("Введіть текст:");
            teststring = in.readLine();
            System.out.println("Зашифрований текст: " + teststring);
            System.out.println("Текст у байтах: "
                    + bytesToString(teststring.getBytes()));
            byte[] encrypted = rsa.encrypt(teststring.getBytes());
            byte[] decrypted = rsa.decrypt(encrypted);
            System.out.println("Розшифрований текст у байтах: " +
bytesToString(decrypted));
            System.out.println("Розшифрований текст: " + new String(decrypted));
        }

        private static String bytesToString(byte[] encrypted)
        {
            String test = "";
            for (byte b : encrypted)
            {
                test += Byte.toString(b);
            }
            return test;
        }

        public byte[] encrypt(byte[] message)
        {
```

```
            return (new BigInteger(message)).modPow(e, N).toByteArray();
        }

        public byte[] decrypt(byte[] message)
        {
            return (new BigInteger(message)).modPow(d, N).toByteArray();
        }
    }public class Signing {

        public static String createSignature(User user, String text){
            try {
                byte[] data = text.getBytes("UTF8");
                Signature sig = user.getSignature();
                sig.initSign(user.getKeyPair().getPrivate());
                sig.update(data);
                byte[] signatureBytes = sig.sign();
                return new BASE64Encoder().encode(signatureBytes);
            }
            catch(Exception e){
                System.out.println(e.getMessage());
                return null;
            }
        }

        public static boolean verifySignature(User user, String signature, String
text){
            try {
                byte[]              signatureBytes              =              new
BASE64Decoder().decodeBuffer(signature);
                byte[] data = text.getBytes("UTF8");
                Signature sig = user.getSignature();
                sig.initVerify(user.getKeyPair().getPublic());
                sig.update(data);
                return sig.verify(signatureBytes);
            }
            catch(Exception e){
                System.out.println(e.getMessage());
                return false;
            }
        }
```

ДОДАТОК В

Система захисту даних на базі методу шифрування RSA.

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ51168_19Б 13-1

Аркушів 8

Київ 2018

АНОТАЦІЯ

Розділ містить опис частини, яка слугує для роботи з БД, що є структурною одиницею програмного продукту, та забезпечує поєднання можливостей усіх інших модулів для виконання поставлених перед системою завдань. Призначенням БД є зберігання інформації. Модуль надає можливість отримувати необхідні дані, оновлювати та видаляти необхідні записи. Модуль написано мовою програмування C#, з використанням технології MS SQL Server.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку розглядається програмний компонент системи, що реалізує алгоритм шифрування RSA, доповнений для реалізації ЕЦП. Компонент здійснює шифування даного повідомлення публічним ключем користувача, генеруючи ЕЦП для цього документу. Друга частина компоненту здійснює дешифрування та перевірку оброблених даних. Результат цієї перевірки говорить, був підписаний документ конкретним користувачем чи ні.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Призначення полягає в тому, щоб забезпечити функціонування електронного документообігу, при цьому захищене за допомогою алгоритмів RSA та ЕЦП. Суть в шифруванні обраного користувачем документу публічним ключем (генерування ЕЦП) та перевірки ЕЦП на приналежність конкретному користувачу. Модуль шифрування повинен забезпечити цілісність та захист від спроб захопити оброблювані дані.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Модуль шифрування використовує алгоритм RSA, генерує ЕЦП, який додається в кінець хешу документу. При перевірці відпрацьовує алгоритм дешифрування, який порівнюючи генеровані підписи, визначає, відповідають вони конкретному користувачу чи ні. Графічний інтерфейс відображає відповідне повідомлення користувачу.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Програмне забезпечення реалізоване як клієнт-серверний додаток, який забезпечує функціонування клієнтської частини та бізнес-логіки окремо один від одного.

Для використання програми потрібен тільки актуальна версія браузера, оскільки саме браузер виступає в ролі клієнта.

Використані методи взаємодіють з даними, розташованими в json файлі.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Модуль розроблено у середовищі розробки IntelliJ IDEA, що забезпечує набір сервісних функцій та широкий спектр модулів для розробки серверної частини додатку. Мова програмування Java – оптимальне та розпоширене рішення для вирішення подібних задач. HTML/CSS використані для створення графічного інтерфейсу користувача. В якості сховища даних використовується файл json формату, дані в якому серіалізовані. Алгоритм RSA використовує вбудований клас BitInteger для коректного множення значень великої довжини.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для програми є інформація, яку користувач вводить в додатку: логін, пароль, обраний для підпису документ.

Вихідними даними програмного модуля є документ, який пройшов алгоритм підпису та інформація про підтвердження або спростування того, чи був документ підписаний.